Outline

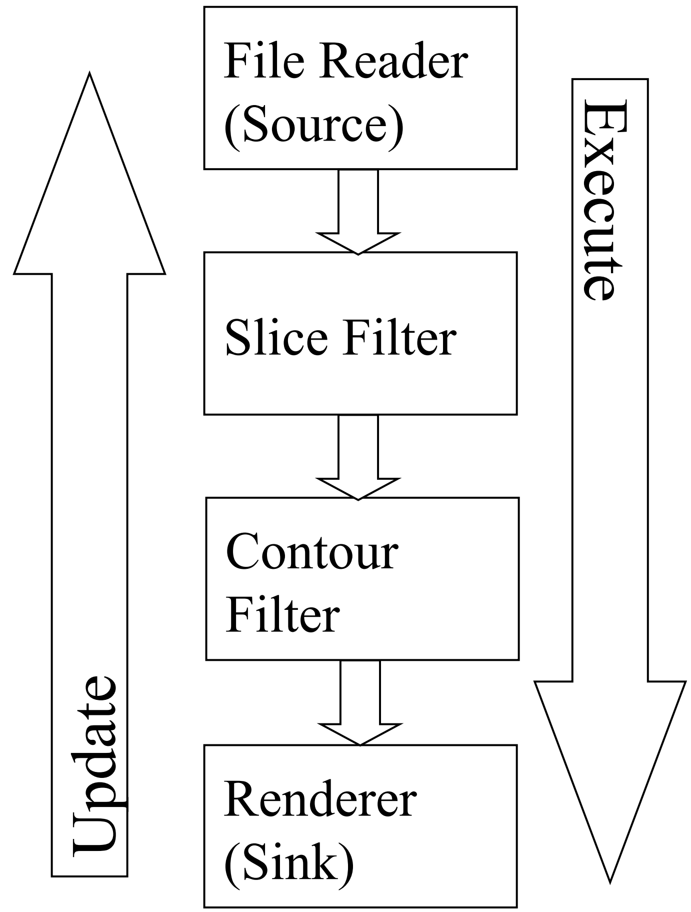
- Data flow networks 101 (3 slides)
- VTK (20 slides)
- Contracts (10 slides)
- An architecture for big data (3 slides)

Outline

- Data flow networks 101 (3 slides)
- VTK (20 slides)
- Contracts (10 slides)
- An architecture for big data (3 slides)

Data flow networks 101

- Work is performed by a *pipeline*
- A pipeline consists of *data objects* and *components* (*sources*, *filters*, and *sinks*)
- Pipeline execution begins with a “pull”, which starts *Update* phase
- Data flows from component to component during the *Execute* phase



Data flow networks: plusses

- Interoperability / Flexibility
- Extensible

Data flow networks: minuses

- Memory efficiency
- Performance efficiency
- Easy to add new algorithms, but hard to extend the data model

Outline

- Data flow networks 101 (3 slides)
- VTK (20 slides)
- Contracts (10 slides)
- An architecture for big data (3 slides)

Visualization with VTK

Content from: Erik Vidholm, Univ of Uppsala, Sweden
David Gobbi, Robarts Research Institute, London, Ontario, Canada

Outline

- What is VTK?
- What can it be used for?
- How do I actually use it?

VTK – The Visualization ToolKit

- Open source, freely available software for 3D computer graphics, image processing, and visualization
- Managed by Kitware Inc.
- Use C++, Tcl/Tk, Python, Java

True visualization system

- Visualization techniques for visualizing
  - Scalar fields
  - Vector fields
  - Tensor fields
- Polygon reduction
- Mesh smoothing
- Image processing
- Your own algorithms

Additional features

- Parallel support (message passing, multithreading)
- Stereo support
- Integrates easily with Motif, Qt, Tcl/Tk, Python/Tk, X11, Windows, ...
- Event handling
- 3D widgets

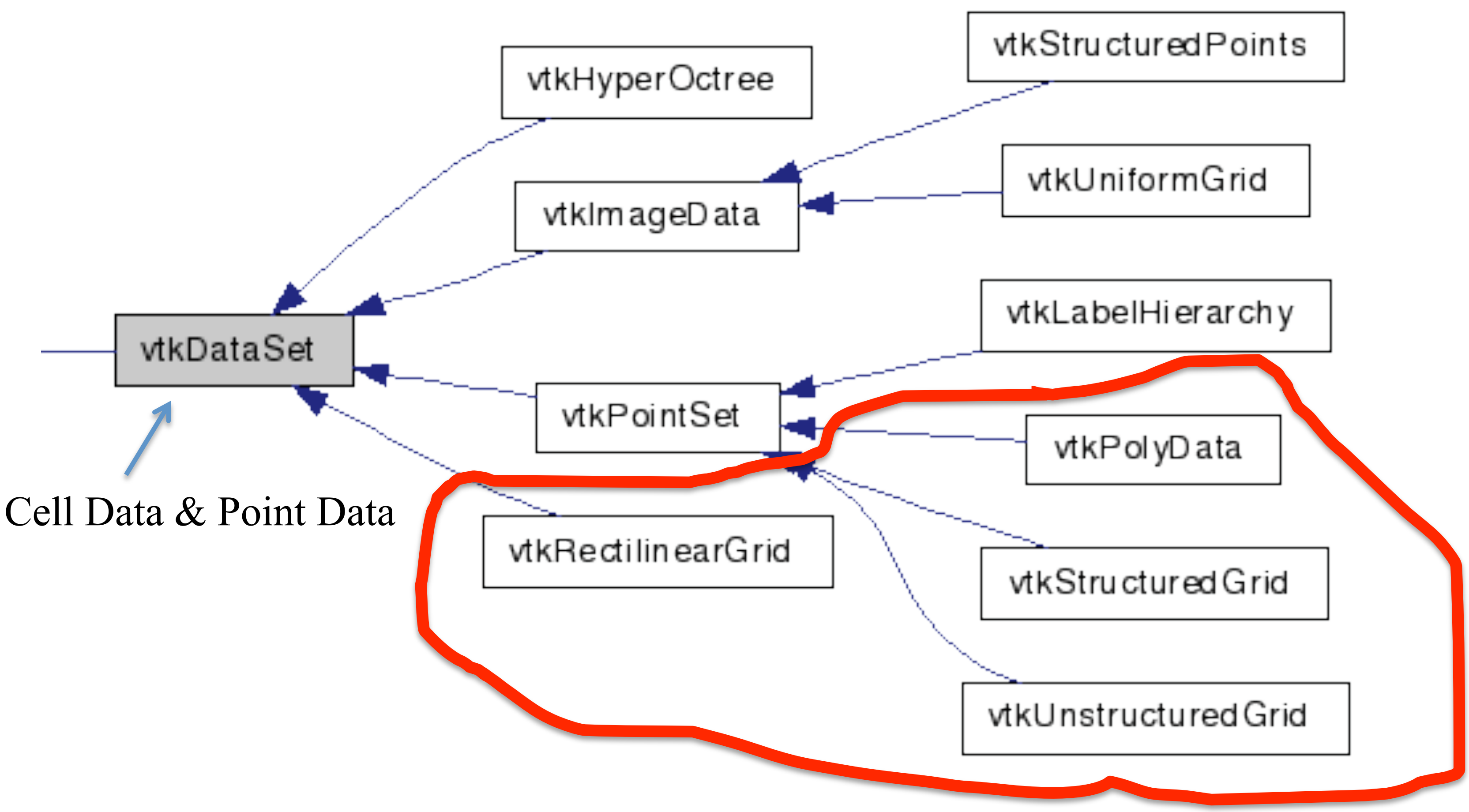
3D graphics

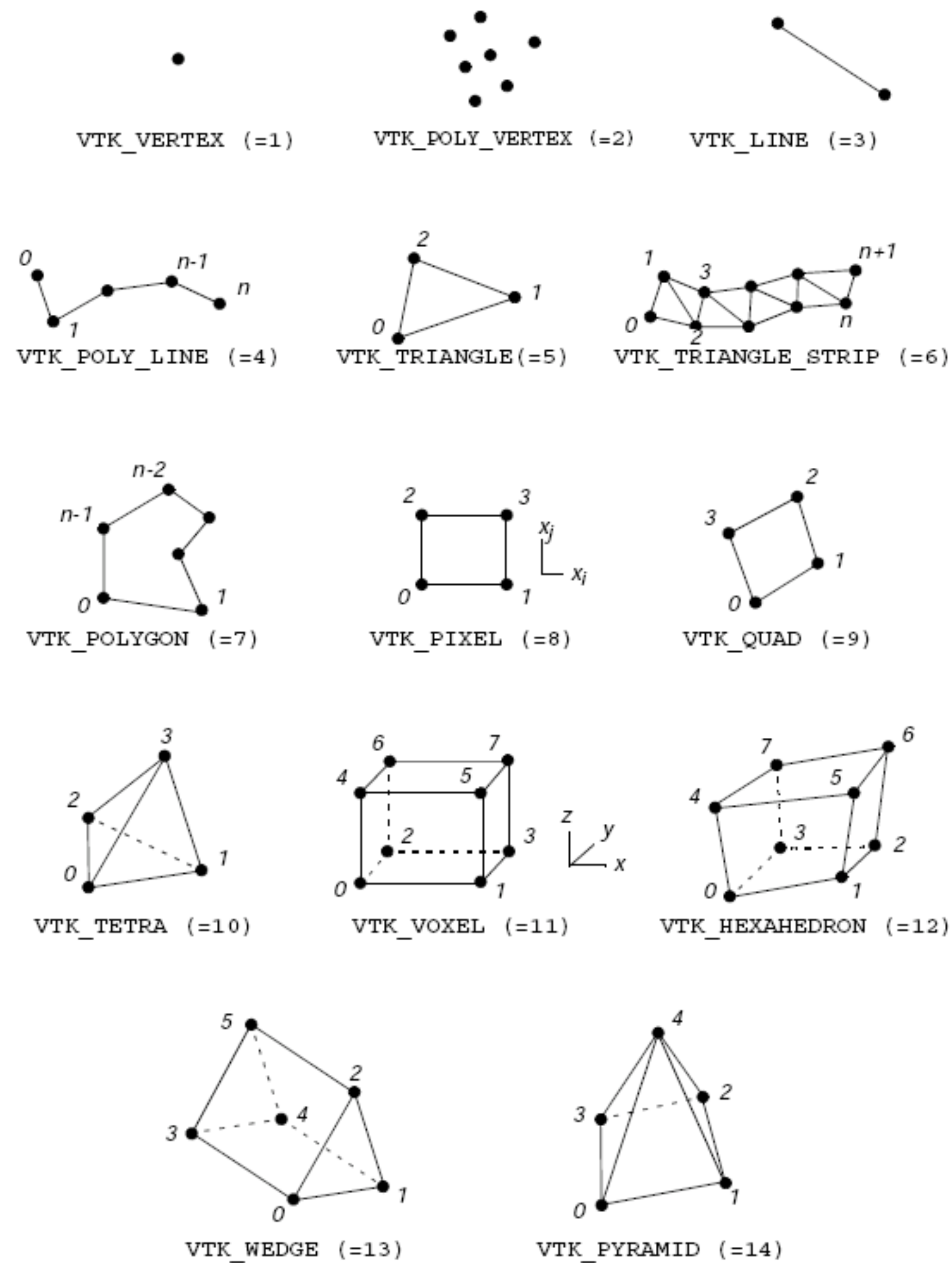
- Surface rendering
- Volume rendering
  - Ray casting
  - Texture mapping (2D)
  - Volume pro support
- Lights and cameras
- Textures
- Save render window to .png, .jpg, ... (useful for movie creation)

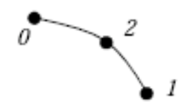
Objects

- Data objects
  - Next slide
- Process objects
  - Source objects (vtkReader, vtkSphereSource)
  - Filter objects (vtkContourFilter)
  - Mapper objects (vtkPolyDataMapper)

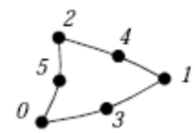
Data model


vtkHyperOctree
vtkStructuredPoints
vtkUniformGrid
vtkImageData
vtkLabelHierarchy
vtkDataSet
vtkPointSet
vtkPolyData
Cell Data & Point Data
vtkRectilinearGrid
vtkStructuredGrid
vtkUnstructuredGrid

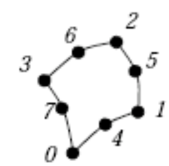

VTK_VERTEX (=1)
VTK_POLY_VERTEX (=2)
VTK_LINE (=3)
VTK_POLY_LINE (=4)
VTK_TRIANGLE(=5)
VTK_TRIANGLE_STRIP (=6)
VTK_POLYGON (=7)
VTK_PIXEL (=8)
VTK_QUAD (=9)
VTK_TETRA (=10)
VTK_VOXEL (=11)
VTK_HEXAHEDRON (=12)
VTK_WEDGE (=13)
VTK_PYRAMID (=14)



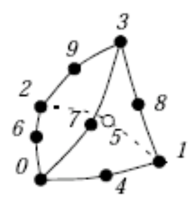
VTK_QUADRATIC_EDGE
(=21)



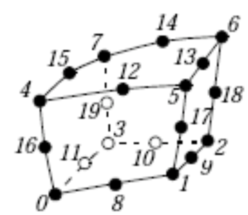
VTK_QUADRATIC_TRIANGLE
(=22)



VTK_QUADRATIC_QUAD
(=23)



VTK_QUADRATIC_TETRA
(=24)

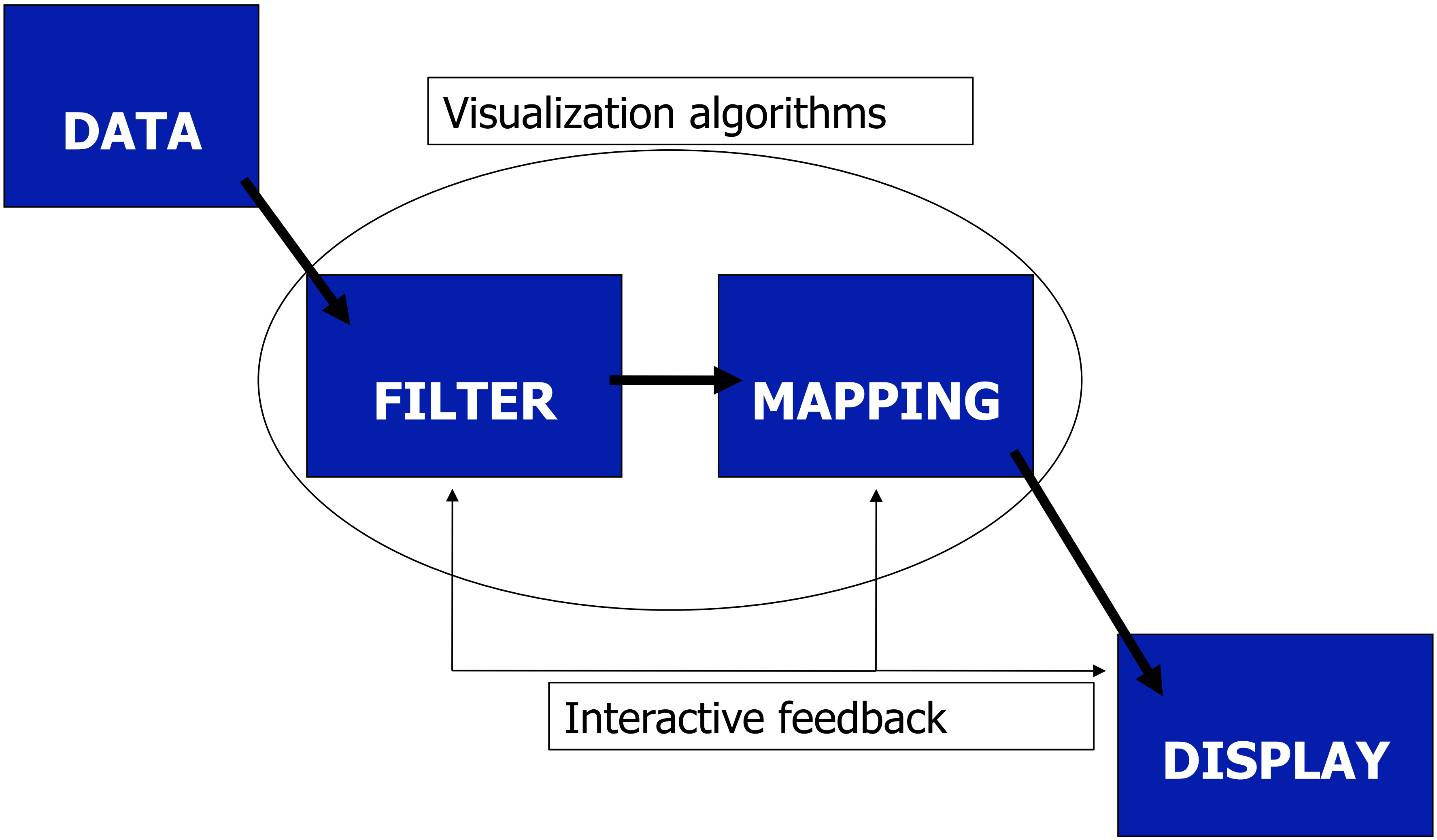


VTK_QUADRATIC_HEXAHEDRON
(=25)

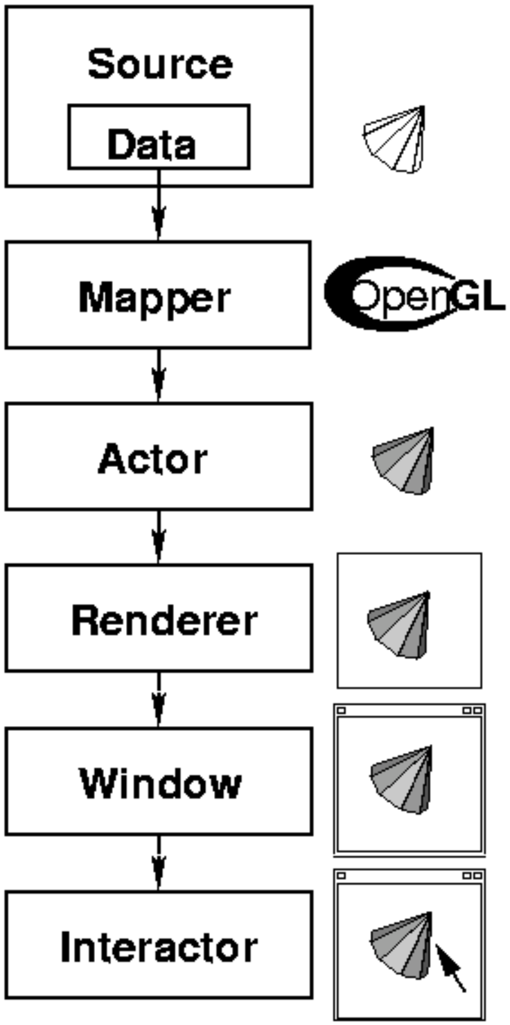
Visualization continued

- Scalar algorithms
  – Iso-contouring
  – Color mapping
- Vector algorithms
  – Hedgehogs
  – Streamlines / streamtubes
- Tensor algorithms
  – Tensor ellipsoids

The visualization pipeline



Cone.py Pipeline Diagram (type "python Cone.py" to run)




```
from vtkpython import *
```

Either reads the data from a file or creates the data from scratch.

```
cone = vtkConeSource()
cone.SetResolution(10)
```

Moves the data from VTK into OpenGL.

```
coneMapper = vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())
```

For setting colors, surface properties, and the position of the object.

```
coneActor = vtkActor()
coneActor.SetMapper(coneMapper)
```

The rectangle of the computer screen that VTK draws into.

```
ren = vtkRenderer()
ren.AddActor(coneActor)
```

The window, including title bar and decorations.

```
renWin = vtkRenderWindow()
renWin.SetWindowName("Cone")
renWin.SetSize(300,300)
renWin.AddRenderer(ren)
```

Allows the mouse to be used to interact wth the data.

```
iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
```

Imaging

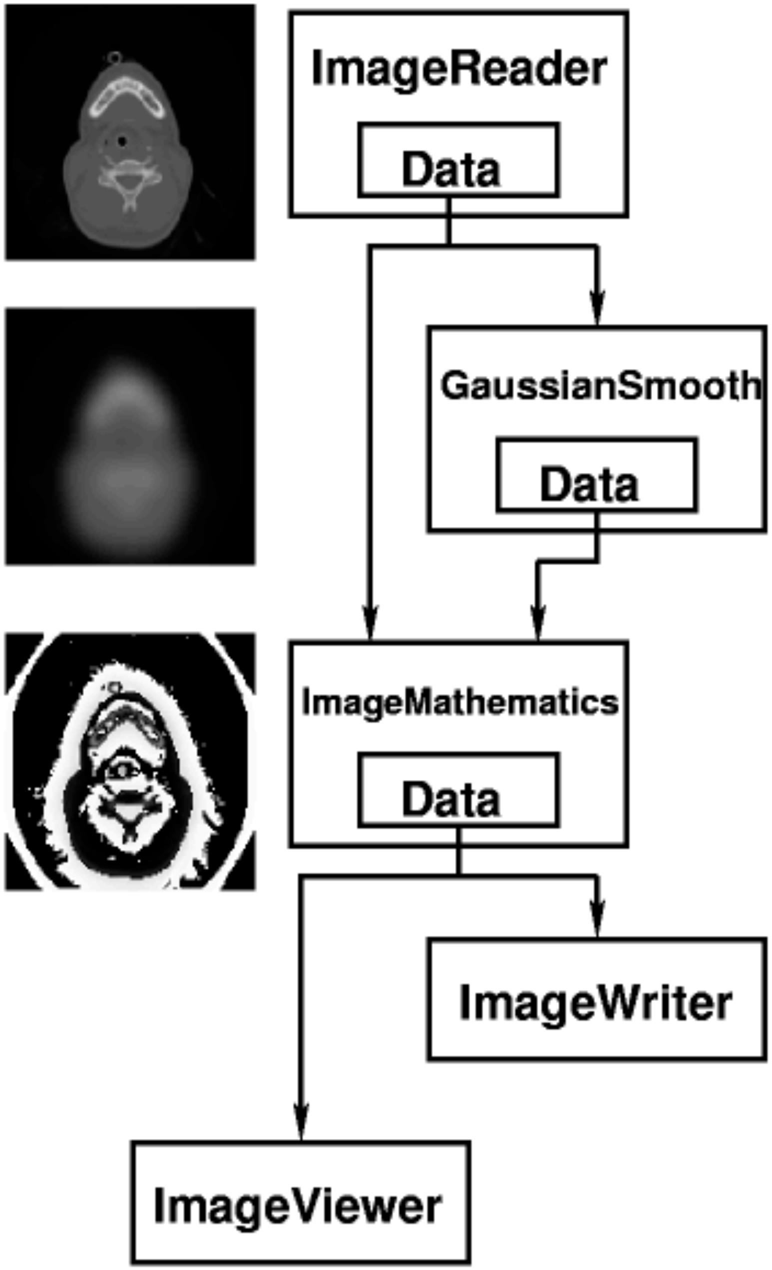
- Supports streaming => huge datasets
- vtkImageToImageFilter
  - Diffusion
  - High-pass / Low-pass (Fourier)
  - Convolution
  - Gradient (magnitude)
  - Distance map
  - Morphology
  - Skeletons

Summary +

- Free and open source
- Create graphics/visualization applications fairly fast
- Object oriented - easy to derive new classes
- Build applications using "interpretive" languages Tcl, Python, and Java
- Many (state of the art) algorithms
- Heavily tested in real-world applications
- Large user base provides decent support
- Commercial support and consulting available

Summary -

- Not a super-fast graphics engine due to portability and C++ dynamic binding – you need a decent workstation
- Very large class hierarchy => learning threshold might be steep
- Many subtleties in usage
  - Pipeline execution model
  - Memory management

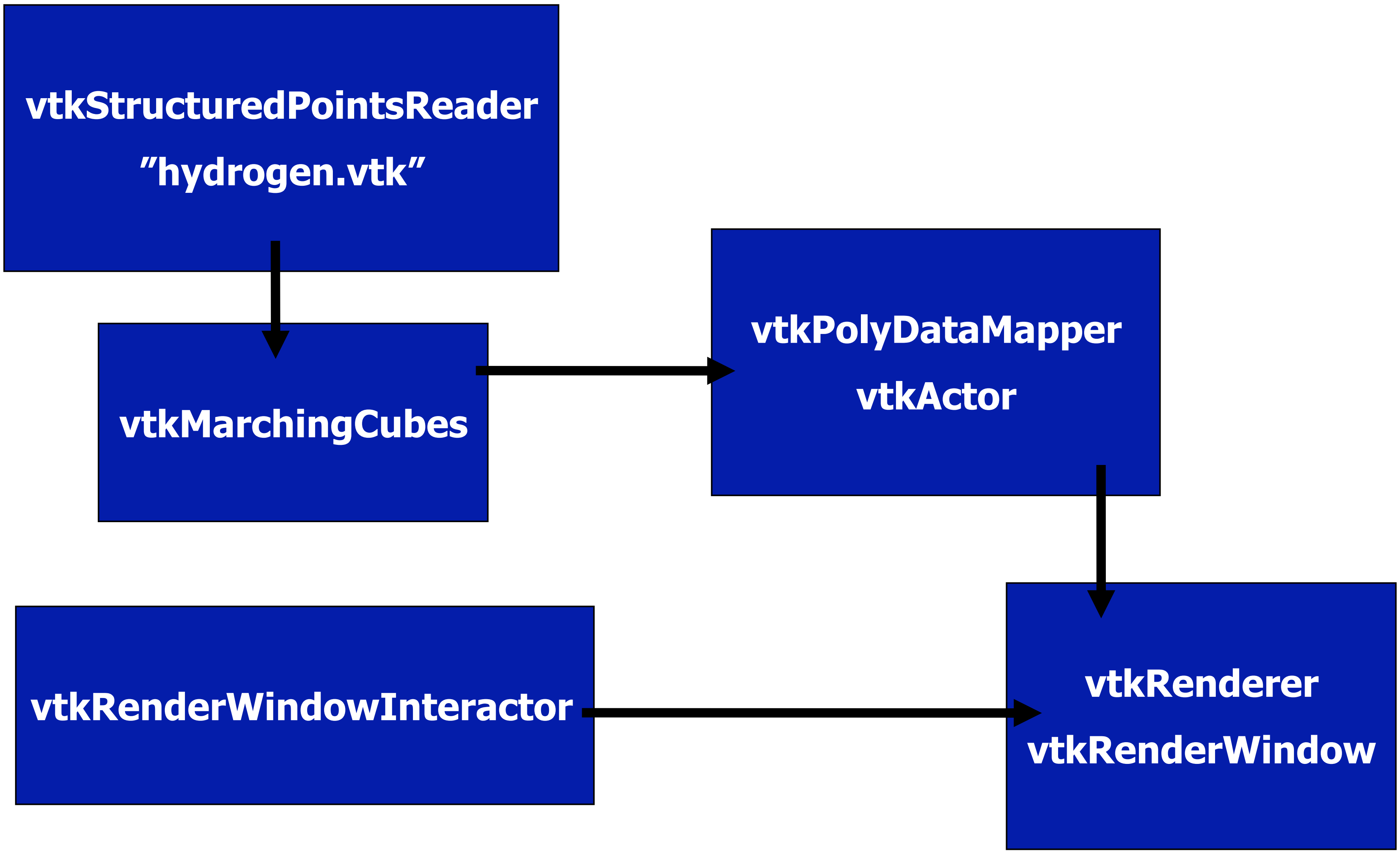



```
reader = vtkBMPReader()
reader.SetFileName("image.bmp")

blur = vtkImageGuassianSmooth()
blur.SetInput(reader.GetOutput())
blur.SetDimensionality(2)
blur.SetStandardDeviations(5.0, 5.0)
blur.SetRadiusFactors(10.0, 10.0)

subtract = vtkImageMathematics()
subtract.SetOperationToSubtract()
subtract.SetInput1(reader.GetOutput())
subtract.SetInput2(blur.GetOutput())

writer = vtkBMPWriter()
writer.SetInput(subtract.GetOutput()
writer.SetFileName("image2.bmp")
writer.Write()

viewer = vtkImageViewer()
viewer.SetInput(subtract.GetOutput())
viewer.SetColorWindow(255)
viewer.SetColorLevel(127.5)
viewer.Render()
```

Example – Vector field visualization

```
vtkStructuredGridReader reader
   reader SetFileName "office.binary.vtk“

# Create source for streamtubes
vtkPointSource seeds
   seeds SetRadius 0.15
   eval seeds SetCenter 0.1 2.1 0.5
   seeds SetNumberOfPoints 6
vtkRungeKutta4 integ
vtkStreamLine streamer
   streamer SetInput [reader GetOutput]
   streamer SetSource [seeds GetOutput]
   streamer SetMaximumPropagationTime 500
   streamer SetStepLength 0.5
   streamer SetIntegrationStepLength 0.05
   streamer SetIntegrationDirectionToIntegrateBothDirections
   streamer SetIntegrator integ
   …
```

The visualization pipeline - example



Python example: visualization hydrogen molecule

Must call update to read!

Pipeline connections

```
# File: isosurface.py
import vtk


# image reader
reader = vtk.vtkStructuredPointsReader()
reader.SetFileName("hydrogen.vtk")
reader.Update()


# bounding box
outline = vtk.vtkOutlineFilter()
outline.SetInput( reader.GetOutput() )
outlineMapper = vtk.vtkPolyDataMapper()
outlineMapper.SetInput( outline.GetOutput() )
outlineActor = vtk.vtkActor()
outlineActor.SetMapper( outlineMapper )
outlineActor.GetProperty().SetColor(0.0,0.0,1.0)
```

Example continued

vtkContourFilter chooses the appropriate method for the data set

```
# iso surface
isosurface = vtk.vtkContourFilter()
isosurface.SetInput( reader.GetOutput() )
isosurface.SetValue( 0, .2 )
isosurfaceMapper = vtk.vtkPolyDataMapper()
isosurfaceMapper.SetInput( isosurface.GetOutput() )
isosurfaceMapper.SetColorModeToMapScalars()
isosurfaceActor = vtk.vtkActor()
isosurfaceActor.SetMapper( isosurfaceMapper )

# slice plane
plane = vtk.vtkImageDataGeometryFilter()
plane.SetInput( reader.GetOutput() )
planeMapper = vtk.vtkPolyDataMapper()
planeMapper.SetInput( plane.GetOutput() )
planeActor = vtk.vtkActor()
planeActor.SetMapper( planeMapper )
```

Example continued

Creates a legend from the data and a lookup table

```
# a colorbar
scalarBar = vtk.vtkScalarBarActor()
scalarBar.SetTitle("Iso value")

# renderer and render window
ren = vtk.vtkRenderer()
ren.SetBackground(.8, .8, .8)
renWin = vtk.vtkRenderWindow()
renWin.SetSize( 400, 400 )
renWin.AddRenderer( ren )
```

Example continued

The RenderWindowInteractor contains functions for mouse/keyboard interaction

```
# render window interactor
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow( renWin )

# add the actors
ren.AddActor( outlineActor )
ren.AddActor( isosurfaceActor )
ren.AddActor( planeActor )
ren.AddActor( scalarBar )

# this causes the pipeline to "execute"
renWin.Render()

# initialize and start the interactor
iren.Initialize()
iren.Start()
```

The renWin.Render() calls Update() on the renderer, which calls Update() for all its actors, which calls…

The VTK file format

- Many modules to write VTK files

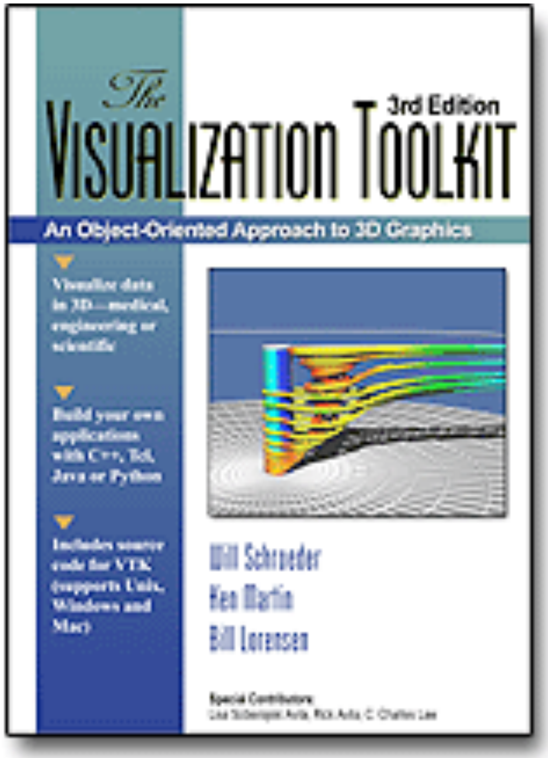
```
# vtk DataFile Version 2.0
Hydrogen orbital
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 64 64 64
ORIGIN 32.5 32.5 32.5
SPACING 1.0 1.0 1.0
POINT_DATA 262144

SCALARS probability float
LOOKUP_TABLE default
0.0 0.0 0.01 0.01 .....
```

VTK and C++

- Build with CMake and your favorite compiler
- CMake generates makefiles or project files for your environment
- Use the resulting file(s) to build your executable
- With C++ you have full control and can derive own classes, but you need to write many lines of code…

VTK resources

- ww.vtk.org
  - Download (source and binaries)
  - Documentation
  - Mailing lists
  - Links
  - FAQ, Search
- ww.kitware.com
  - VTK Textbook
  - VTK User's guide
  - Mastering CMake



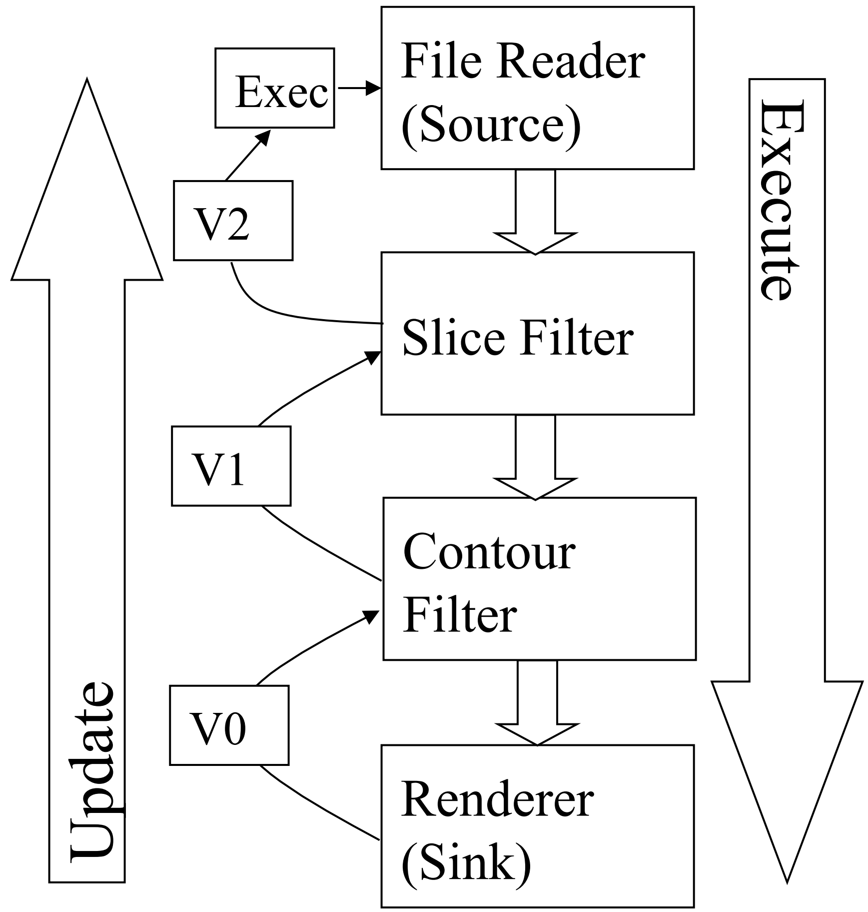
Outline

- Data flow networks 101 (3 slides)
- VTK (20 slides)
- Contracts (10 slides)
- An architecture for big data (3 slides)

Contracts are an extension to the standard data flow network design.

Data Flow Networks "101":

- Work is performed by a *pipeline*
- A pipeline consists of *data objects* and *components* (*sources*, *filters*, and *sinks*)
- Pipeline execution begins with a "pull", which starts *Update* phase
- Data flows from component to component during the *Execute* phase



Extension:

- Contracts are coupled with the *Update* phase

Initial observations about contracts.

- A contract is simply a data structure
  - The members of the data structure reflect the optimizations
- Optimizations are adaptively applied based on the final contract
- Each component describes its *impact* on the pipeline
  - Allows for effective management of hundreds of components
  - Allows for new, unforeseen components to be added
- Combining contracts with the Update phase

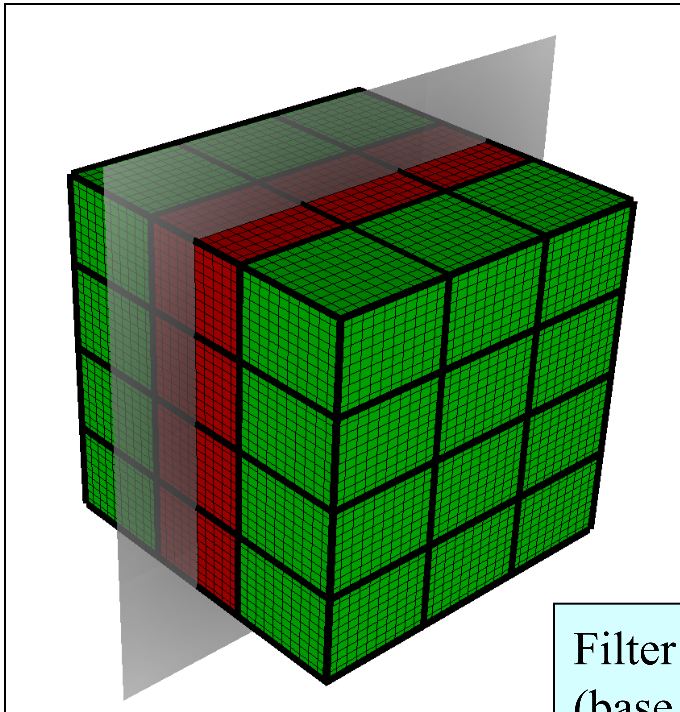
  →seamlessly integrated into data flow networks

  →every component has a chance to modify the contract

Why are contracts important?

1) They allow for optimizations to be utilized in a richly featured system
2) Is this important? Yes. Let's look at the impact of these optimizations. (If we believe they are important, then contracts are important.)

Operating on Optimal Subset of Data



1) Get meta-data
2) Determine domains that intersect slice
3) Restrict list of domains to process in V(I+1)

Filter
(base class)

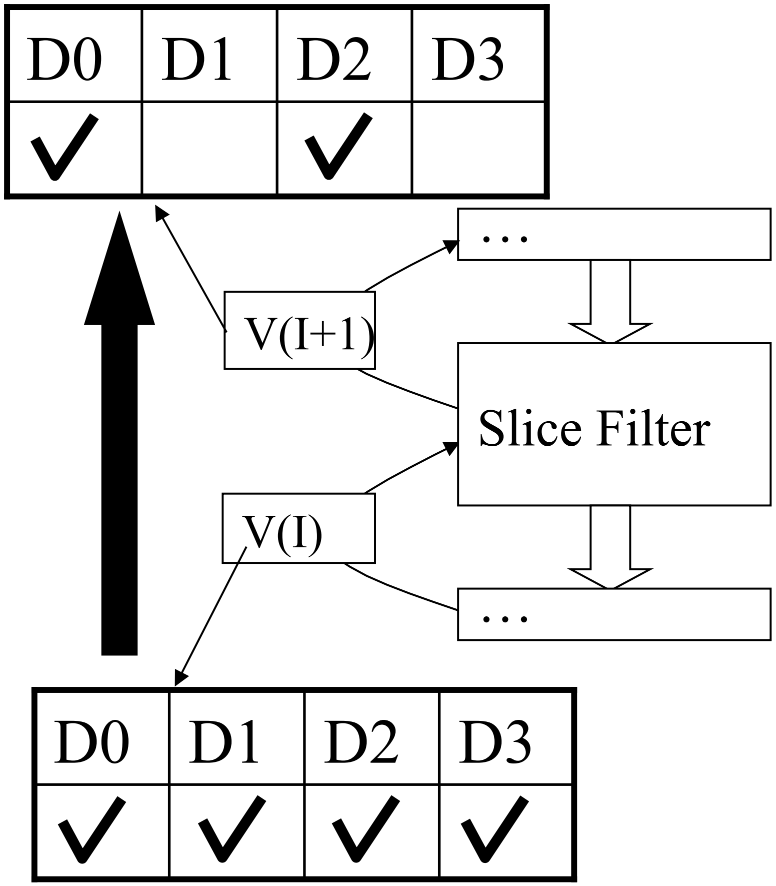
(inheritance)

Slice Filter

Hundreds
of others

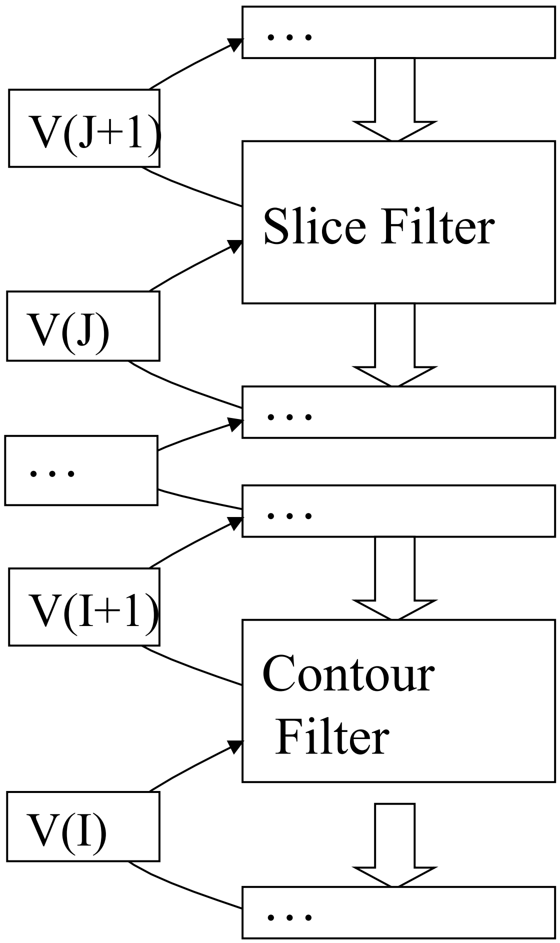
```
...

ontract
          (Contract *V(I))
{
     ...
     return V(I+1);
}
```

Operating on Optimal Subset of Data

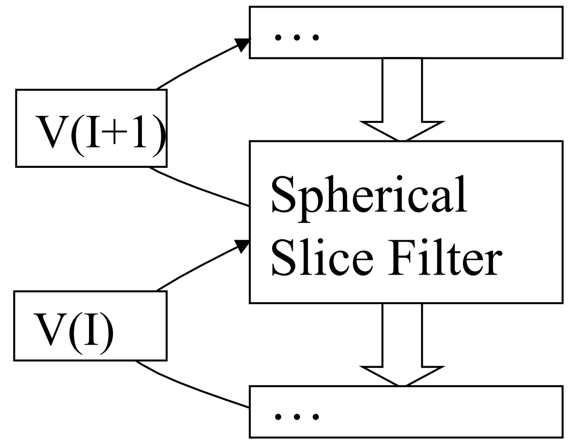


1) Get meta-data
2) Determine domains that intersect slice
3) Restrict list of domains to process in V(I+1)

The contract-based system provides high flexibility for this optimization.



Multiple filters can use the same optimizations



A new, plugin filter can use this optimization without any modification to VisIt proper

We studied performance using a simulation of a Rayleigh-Taylor Instability.

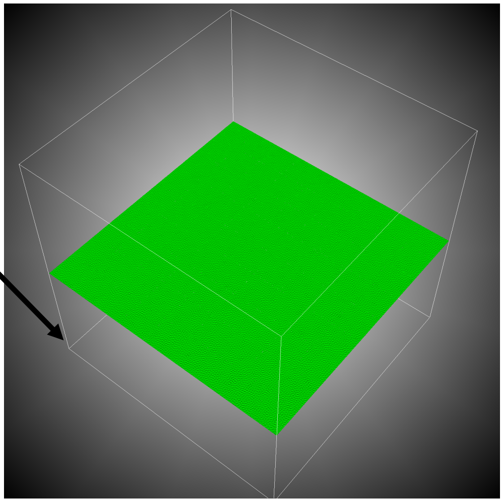
The techniques shown are not new

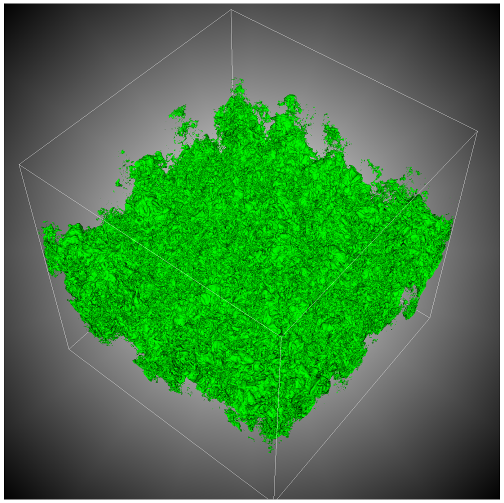
The performance increase motivates the importance of optimizations

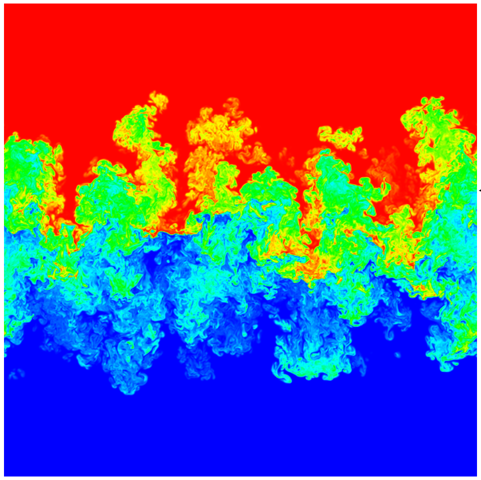
This, in turn, motivates the importance of contracts

Processing only the necessary domains is a lucrative optimization.

Algorithm	Processors	Without Contracts	With Contracts	Speedup
Contouring (early)	32	41.1s	5.8s	7.1X
Contouring (late)	32	185.0s	97.2s	1.9X
Slicing	32	25.3s	3.2s	7.9X







What is the right technique for distributing domains across processors?

- Two ways:
  - Statically: make assignments before Execute phase
  - Dynamically: adaptively during Execute phase
- Performance:
  - Static: good chance of load imbalance
    - As fast as slowest processor
  - Dynamic: adaptively balancing load
    - Obtains near optimal parallel efficiency
- Communication:
  - Static: collective communication okay
  - Dynamic: no collective communication

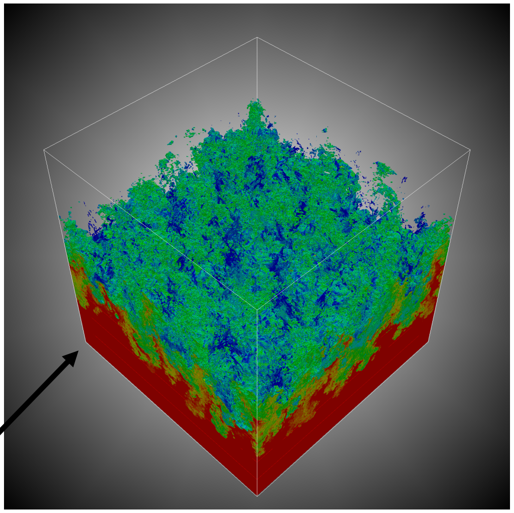
Contracts steer what load balancing technique we use.

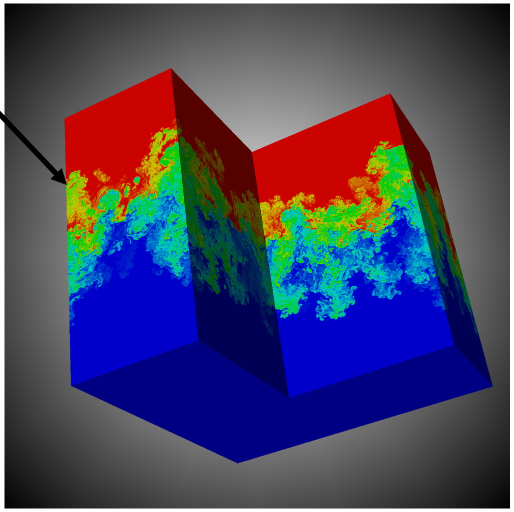
- What load balancing technique should we use?
- If we need collective communication → static
- Otherwise, we want performance → dynamic

- Contracts enable this
  - During Update phase:
    - Every filter can modify the contract to state whether or not it needs collective communication
  - Before Execute phase:
    - Executive examines contract and decides which load balancing technique to use.

Employing dynamic load balancing is a lucrative optimization.

Algorithm*	Processors	Static Load Balancing	Dynamic Load Balancing	Speedup
Slicing	32	3.2s	4.0s	0.8X
Contouring	32	97.2s	65.1s	1.5X
Thresholding	64	181.3s	64.1s	2.8X
Clipping	64	59.0s	30.7s	1.9X



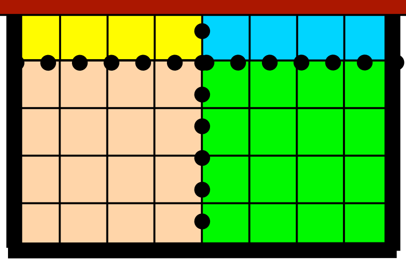


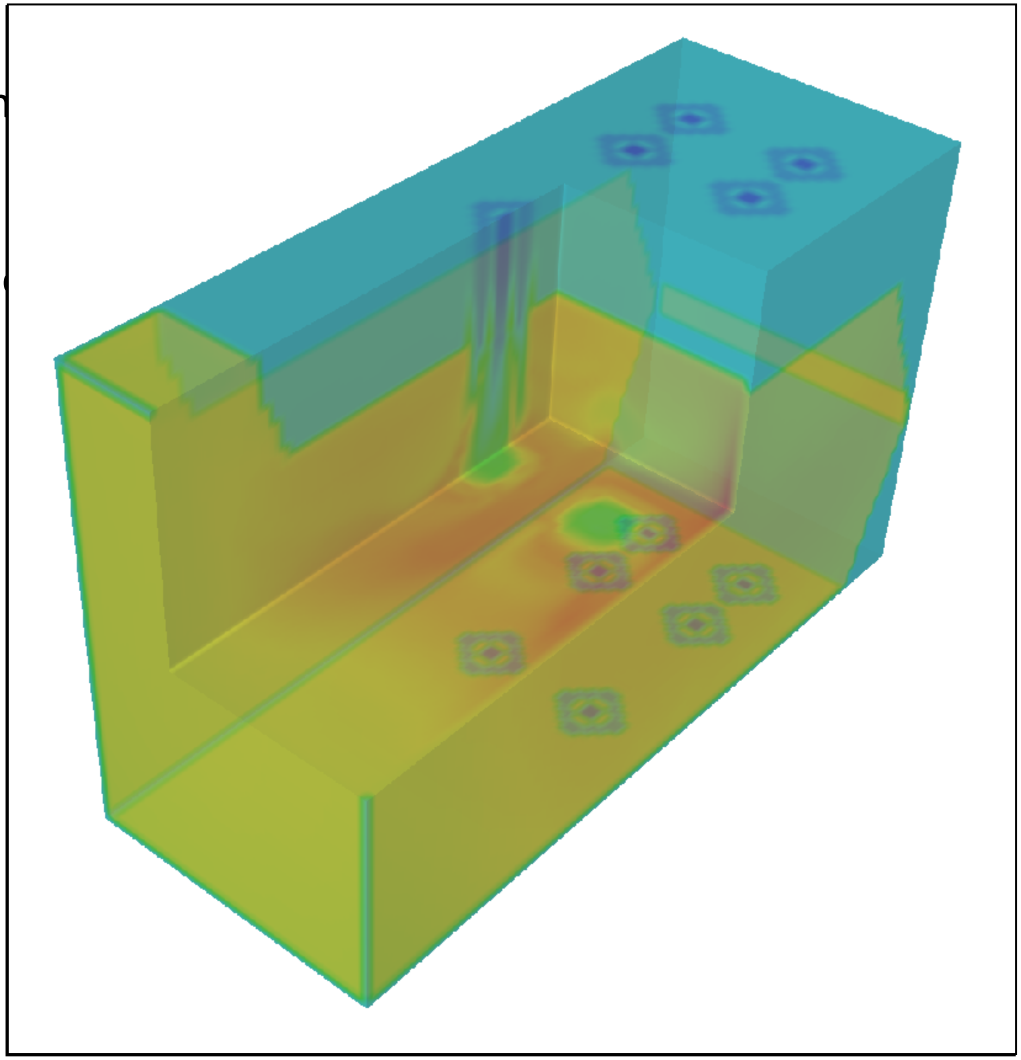
** = All of these operations have no collective communication*

Artifacts occur along the boundaries of domains.

- Looking at external faces
  - Faces external to a domain can
    internal to the data set
  - → many extra, unneeded faces
  - → wrong picture with transparen

Solution: mark unwanted faces as "ghost"



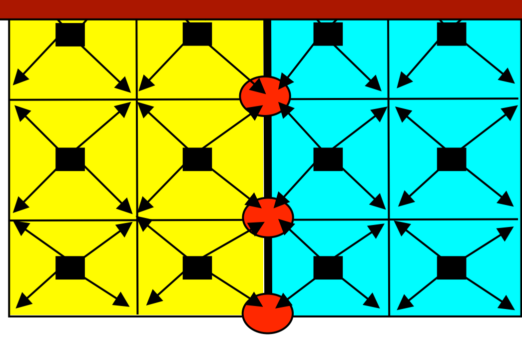


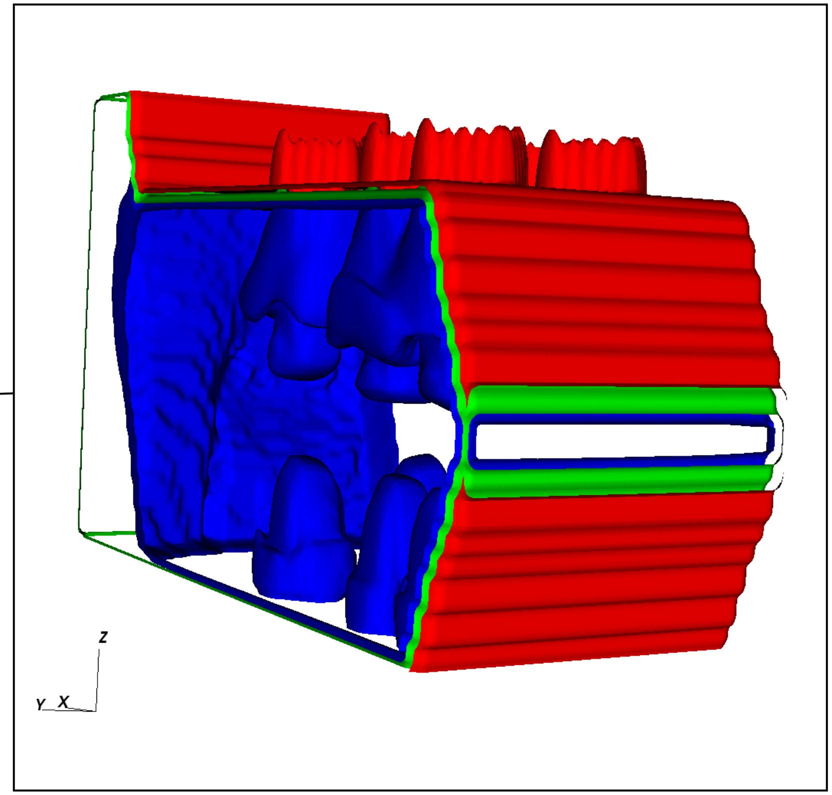
Artifacts occur along the boundaries of domains.

- Interpolation
  - Inconsistent values at nodes along boundary

es

Solution: make redundant layer of "ghost" elements





Ghost data fixes artifacts along domain boundaries.

- VisIt can generate ghost data on the fly

- Through contracts, VisIt determines necessary

We always get the right picture,
and we do it with the minimum cost

- There are different costs for ghost data:
  - Ghost faces: memory
  - Ghost elements: memory, collective communication

Contracts are a simple idea that have a large impact.

- Contracts:
  - Just a data structure
  - Describe what impact a component has on the pipeline

Name	Type	Default Value
domains	vector<bool>	all true
hasColl-Commun.	bool	false
ghostType	enum {None, Face, Element}	None
much more…	…	…

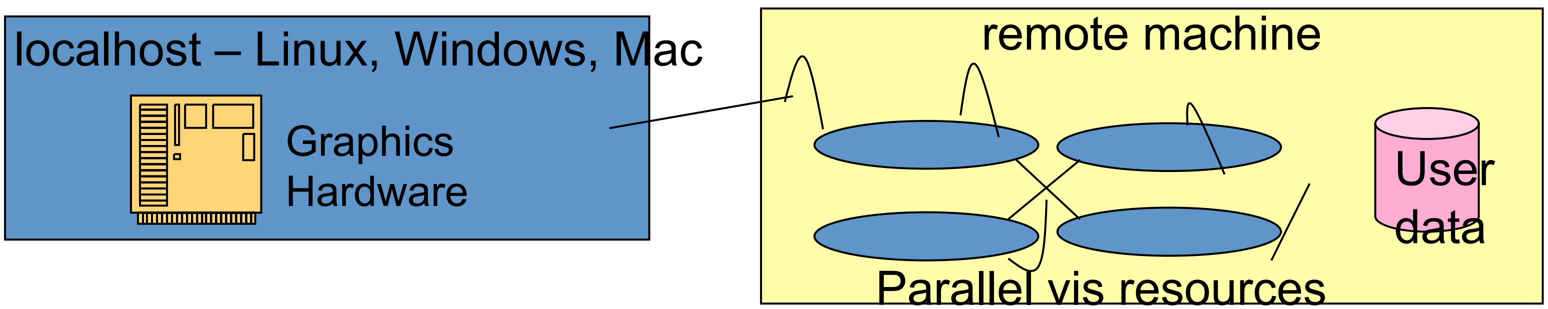
➢ Contracts enable us to avoid the following "dumb" (conservative) strategies:

- Read all data
- Always assume collective communication
- Always create ghost elements

Outline

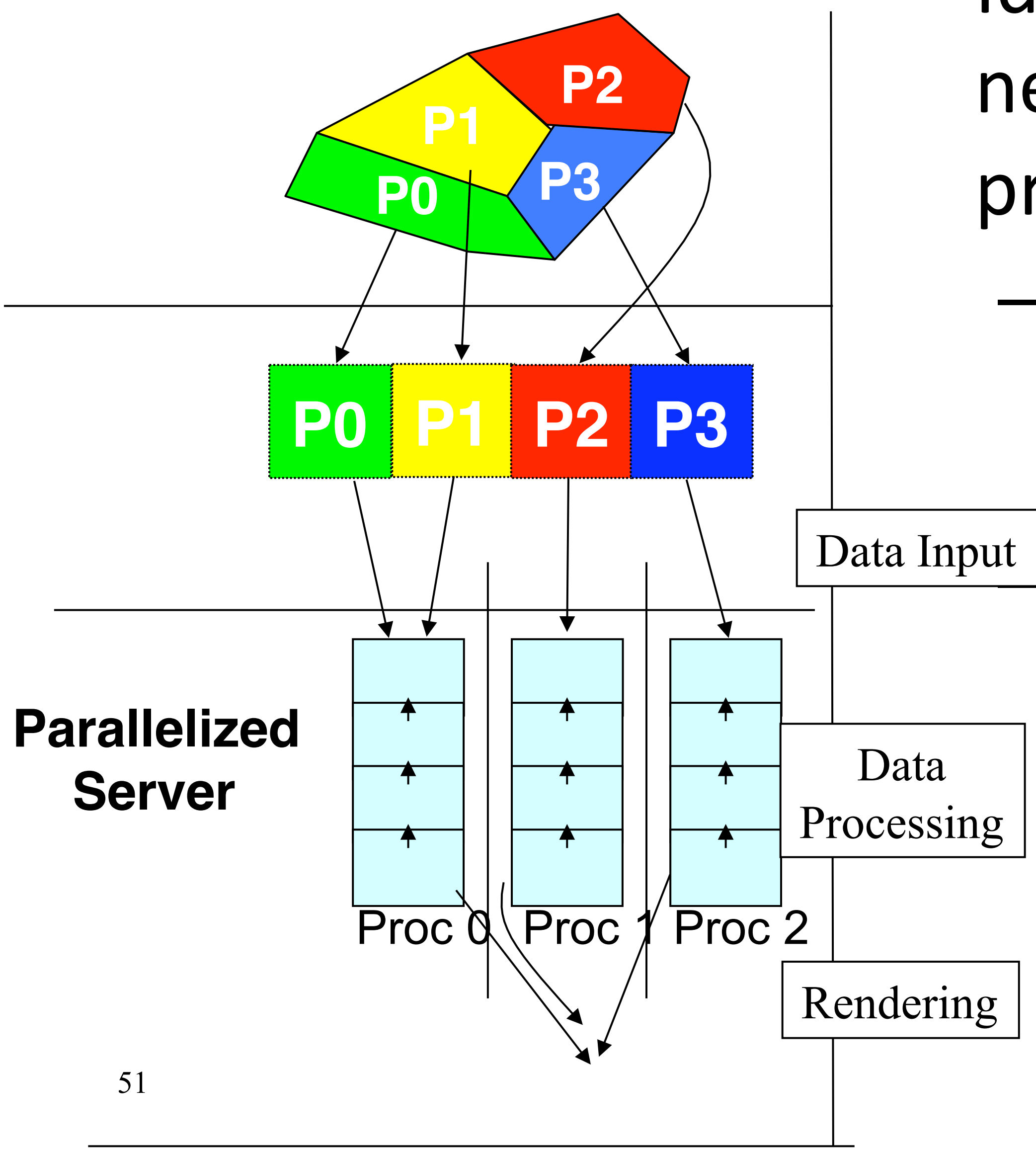
- Data flow networks 101 (3 slides)
- VTK (20 slides)
- Contracts (10 slides)
- An architecture for big data (3 slides)

VisIt employs a parallelized client-server architecture.



- Client-server observations:
  - Good for remote visualization
  - Leverages available resources
  - Scales well
  - No need to move data
- Additional design considerations:
  - Plugins
  - Multiple UIs: GUI (Qt), CLI (Python), more…
  - Third party libraries: VTK, Qt, Python, Mesa, +I/O libs

Parallelization covers data input, data processing, and rendering.



- Identical data flow networks on each processor.
  - Networks differentiated by portion of data they operate on.
  - "Scattered/gather"
    - No distribution (i.e. scatter), because scatter is done with choice of what data to read.
    - Gather: done when rendering